QCQT-QE9 Quantum Machine Learning 2024-2025

Prof. I. Mpoutalis - Topalidis Dimitrios (Registration number 60687)
2025 Questions - Exercises for Unsupervised ML

Question 1

Describe the basic idea of Unsupervised Learning and mention the main categories of Unsupervised Learning algorithms

Unsupervised learning is a method of machine learning that finds structures in data. Labels for the data are not necessary because the machine is able to learn on its own without superivision. Unsupervised learning aims at clustering, probability density estimation, finding association among features, and dimensionality reduction.
Thus, the main categories for Unsupervised Learning algorithms are:

- Clustering (e.g. K-means, herarchical clustering, density based clustering etc.)
- Finding association rules
- Dimensionality reduction (e.g. Principal Component Analysis, multidimensional scaling, manifold learning)

Question 2

What is the basic idea of Principal Component Analysis (PCA) technique and where can be used?

The Principal Component Analysis (PCA) technique is used as a dimensionality reduction method. It is one of the simplest approaches to obtain a lower dimensional representation that describes higher dimensional data. There are specific cases that this is particularly useful, such as:

- **Visualization**. PCA can project data to lower dimensions that can be plotted or otherwise efficiently visualized.
- **Data preprocessing**. The number of dimensions, meaning that very large data sets are required for higher-dimensional models. PCA can be used to first map the data to a low-dimensional representation before applying a more sophisticated algorithm to it
- **Compression**. PCA can help compressing data with its lower dimensional representation.
- **Modeling**. PCA learns a representation that is sometimes used as an entire model

Question 3

From the PCA technique, write the equation by which we calculate the low dimensional vector from the original high-dimensional one, explaining what each term of the equation is

Assume we have a set of N D-dimensional data vectors $\mathbf{y_i}$
We want to replace those vectors with C-dimensional vectors $\mathbf{x_i}$ with $C < D$
The equation from which we calculation the low dimensional vector from the high dimensional one is:

$$x_i = W^T(y_i - b)$$

where:

- $W = [V_1, \dots, V_C]$ contains the first C eigenevectors sorted
- x_i as we said is the lower dimensional vector
- y_i as we said is the original higher dimensional vector
- $b = \dfrac{1}{N}\sum_{i=1}^{N} y_i$ is the mean offset

Question 4

What is Manifold Embedding and which is its difference from the PCA technique?

Manifold embedding refers to a broad range of algorithms which assume that data in a high-dimensional space align to a manifold in a space of much lower dimensions.
Its difference with the PCA technique is that it can discover curved or non linear structures in data, while the PCA technique cannot because it is based on a linear transformation.

Question 5

List well -known Manifold Embedding algorithms.

A well known Manifold Embedding algorithm is the **Isomap algorithm (Isometric mapping)**. It aims to approximate the geometry of the data before projecting it down into the specified dimension by maintaining a geodesic distance between two points. It is a computationaly expensive algorithm. There is an extension called **L-isomap** which utilizes landmark selection in order to avoid expensive computations, resulting in improved run time.

Question 6

What is data clustering and where can it find applications? What are the key categories of clustering techniques?

Clustering aims at grouping a set of data objects in such a way that objects in the same group are more similar to each other than to those in other groups. Those groups are called **clusters**.
Clustering finds applciations in many fields and tasks. Some of them are:

- Market research
- Documents clustering
- Working with images
- Finding the underlying structure of complicated data
- Data compression

The main clustering techniques are the following:

- **Hierarchical clustering (connectivity based clustering)**. It is based on the idea that objects are more realated to other objects near them, than the ones that are far away. The algorithms that implement Hierarchical clustering form clusters of objects based on their distance.
- **Centroid based clustering**. This technique involves iterative clsutering algorithms that derive the notion of similarity between data within a cluster by the distance of a data point from the centroid of the clusters. Those algorithms require knowing the number of clusters beforehand thus making prior knwoledge of the dataset mandatory. One popular algorithm that falls in this category is the K-means algorithm.
- **Density based clustering**. Clusters are defined as areas of higher density. Objects in low density areas between the clusters are considered to be noise.
- **Distribution based clustering**. This thechnique is based on the notion of how probable is that all data points within a cluster belong to the same distribution.

Question 7

In which category of clustering techniques do K-Means, K-Medoids, K-Medians belong?

The K-means algorithm and its variants K-medoids and K-medians, all belong to the **Centroid based clustering** category.

Question 8

In a nutshell, describe the Hierarchical clustering technique and the two main approaches (divisive, agglomerative)

As we have already seen, the K-means algorithm is a method to cluster data samples by the distances between the data points, into K clusters, where K must be known beforehand, trying to minimize intracluster variance.
Hierarchical clustering is as simple as K-means, but instead of having a fixed number of clusters, the number changes in every iteration
There are two types fof Hierarchical Clustering:

1. **Divisive (top-down)**. We assign all of the observations to a single cluster and then partition the cluster to two least similar clusters. We proceed recursively on each cluster until there is one cluster for each observation.
2. **Agglomerative (bottom-up)**. We begin with every observation representing a singleton cluster. At each of the following $N-1$ steps the closest two clusters are merged into a single cluster, producing one less cluster at the next higher level. For this to be possible we must define a measure of how close two clusters are (measure of dissimilarity). We compute this measure from the set of pairwise measurements of how close two observations are, where always one observation is from one cluster and one from another.

Question 9

Mention, explaining their differences, the alternative ways of calculating the intergroup dissimilarity used in Agglomerative clustering

Regarding the measure of dissimilarity we talked about in the previous question, that needs to be defined when implementing agglomerative clustering, there are three main approaches of calculating it:

1. **Single Linkage (SL)**, which takes the intergroup dissimilarity to be that of the closest (least dissimilar) pair.
2. **Complete linkage (CL)**, which takes the intergroup dissimilarity to be that of the furthest (most dissimilar) pair.
3. **Group average (GA)**, which uses the average dissimilarity between groups.

Question 10

What is Density Based Clustering? Say briefly how DBSCAN algorithm works

Density based clustering is a clustering technique that identifies clusters based on the idea that a cluster is a region of high data point density and is separated from other clusters by regions of low data point density. Thus, the shapes of the clusters can be arbitrary. The data points in the separating regions of low point density are typically considered as noise or outliers.

The most known density based algorithm is DBSCAN.
It takes two input parameters:

- ϵ, which defines a neighborhood for each data point
- N_{min}, which defines the minimum number of points required to form a dense region.

Then the following definitions are useful:

- **ϵ-neighborhood** of a point is the set of all points with distance ϵ from it, including itself.
- **Core point**. A point that has at least N_{min} points in its ϵ-neighborhood, including itself
- **Border point**. A point that is not a core point, but lies within the ϵ-neighborhood of a core point

- A point that is neither a core point nor a border point is considered **noise**

The algorithm works as follows:

1. For each data point in the dataset, it finds all other points within its ϵ-neighborhood and counts them.
2. Marks all core points (points that have more that N_{min} points in their ϵ-neighborhood)
3. Construcs a graph where each core point is a node and an edge exists between two core points if they are within ϵ-neighborhood of each other. **Connected components in this graph form the initial clusters.**
4. Assigns each non-core point to a cluster, if its within the ϵ-neighborhood of that cluster, otherwise labels it as noise.

Question 11

Indicate the best-known approaches to identify the number of clusters in a data clustering algorithm. Do we need to determine the number of clusters in all data clustering algorithms?

As we saw in previous questions, some clustering techniques do not require the number of clusters K to be known beforehand, like Hierarchical and density based clustering.
Some other techniques though need K to be specified, like like the K-means, K-medoids, K-medians algorithms etc. If this number cannot be specified from prior knowledge of the properties of the data set, the are some other approaches:

- **The Elbow method**. The number of clusters is chosen to minimize the intra-cluster variation (or total within-cluster sum of square (WSS)).
- **The Silhouette method**. The method computes **silhouette coefficients** of each point that measure how much a point is similar to its own cluster compared to other clusters. Then, average it out for all the samples to get the **silhouette score**. The optimal number of clusters K is the one that maximizes the silhouette score over a range of possible values for K.

  The formula for the silhouette coefficients is:

$$s(i) = \frac{b(i) - a(i)}{max(b(i) - a(i))}$$

  with $a(i)$ be the average distance of the i^{th} data point with all other points in the same clusters and $b(i)$ its average distance with all the points in the closest cluster to its cluster.

- **The Gap statistics method**. This method selects the appropriate number of cluster by solving the following minimization problem:

$$K^* = \arg\min_K \big\{K | G(K) \geq G(K+1) - S'_{K+1}\big\}$$

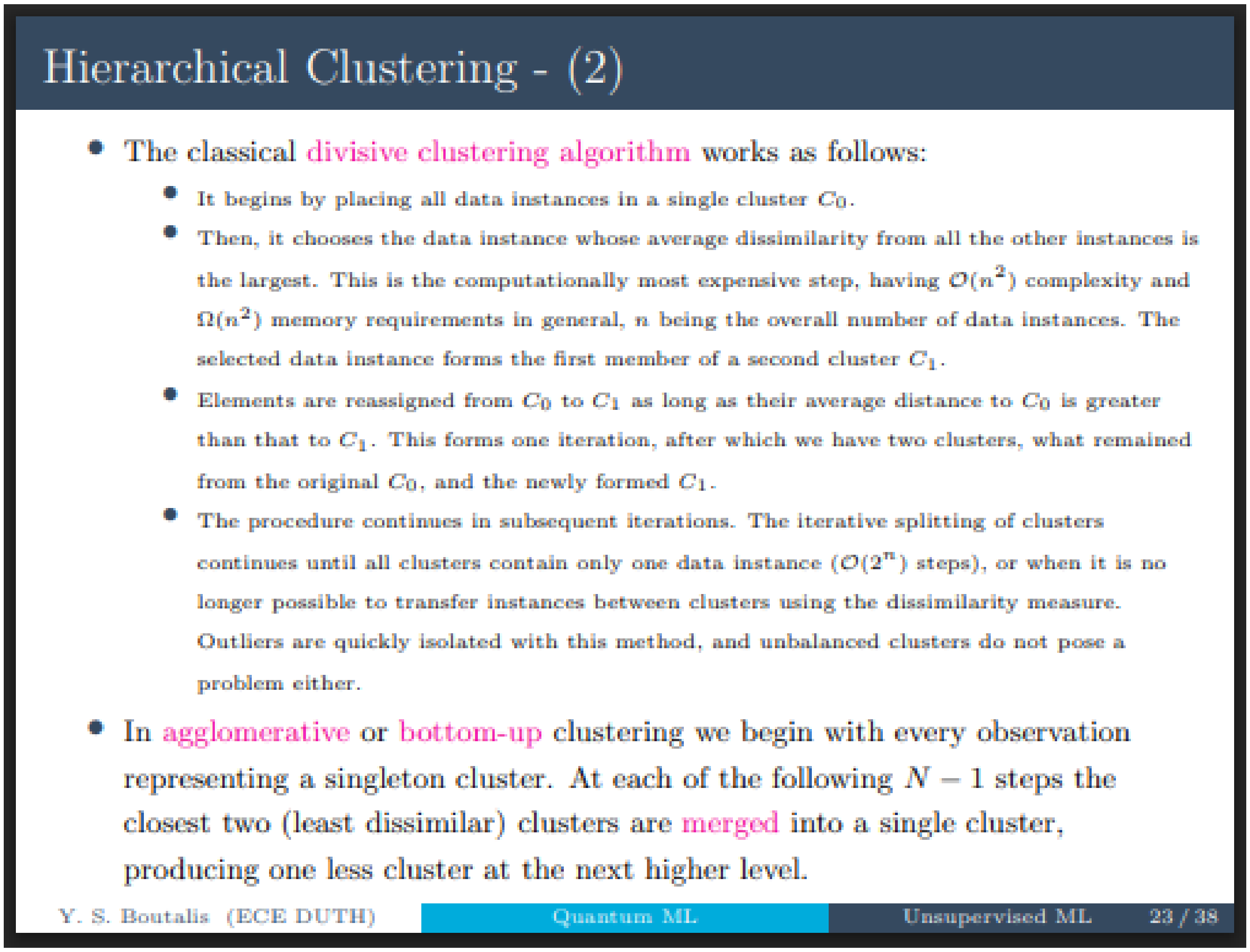
  where:

  - $G(K) = log(W_K^{unif}) - log(W_K^{data})$ where W_K^{unif} and W_K^{data} are the within-cluster sum of square distances from the cluster means of a reference distribution and of the data in our problem, respectively.
  - $S'_{K+1} = S_K\sqrt{1 + \frac{1}{20}}$ where $S_K =$ the standard deviation of W_K^{unif} across 20 runs

Question 12

Implement the divisive clustering technique (for data clustering) in python (Use appropriate data and show graphically or otherwise - the result of applying the technique).

My registration number is 60687 so (60687 mod 6) + 1 = 4. So I must solve the 4th subquestion of question 12. You will find the code of the implementation of the Divisive Clustering technique below.
This is a custom implementation, based on what we saw on the lecture notes. Specifically I was based in the divisive clustering algorithm described in the file **Qml_3-UnsupML.pdf** (`https://eclass.duth.gr/modules/document/file.php/1031590/Qml_3-UnsupML.pdf`), **page 23**, as found in e-Class.



Hierarchical Clustering - (2)

- The classical divisive clustering algorithm works as follows:
  - It begins by placing all data instances in a single cluster C_0.
  - Then, it chooses the data instance whose average dissimilarity from all the other instances is the largest. This is the computationally most expensive step, having $\mathcal{O}(n^2)$ complexity and $\Omega(n^2)$ memory requirements in general, n being the overall number of data instances. The selected data instance forms the first member of a second cluster C_1.
  - Elements are reassigned from C_0 to C_1 as long as their average distance to C_0 is greater than that to C_1. This forms one iteration, after which we have two clusters, what remained from the original C_0, and the newly formed C_1.
  - The procedure continues in subsequent iterations. The iterative splitting of clusters continues until all clusters contain only one data instance ($\mathcal{O}(2^n)$ steps), or when it is no longer possible to transfer instances between clusters using the dissimilarity measure. Outliers are quickly isolated with this method, and unbalanced clusters do not pose a problem either.
- In agglomerative or bottom-up clustering we begin with every observation representing a singleton cluster. At each of the following $N-1$ steps the closest two (least dissimilar) clusters are merged into a single cluster, producing one less cluster at the next higher level.

Y. S. Boutalis (ECE DUTH) Quantum ML Unsupervised ML 23 / 38

For my dataset, I used the files:

- om1.txt (`https://eclass.duth.gr/modules/document/file.php/1031590/Files%20for%20exercises/Unsupervised%20ML/om1.txt`)
- om2.txt (`https://eclass.duth.gr/modules/document/file.php/1031590/Files%20for%20exercises/Unsupervised%20ML/om2.txt`)
- om3.txt (`https://eclass.duth.gr/modules/document/file.php/1031590/Files%20for%20exercises/Unsupervised%20ML/om3.txt`)

as found in e-Class and as advised by the professor.
In order to run the code you also must have python installed in your system as well as the libraries **matplotlib** and **numpy**

You should also place the dataset files in the directory in which you run the code with the filenames **om1.txt**, **om2.txt**, **om3.txt**
You can also find all of the respective code and files in the following repository: `https://github.com/topalidisd-qcqt-duth/qe9-assignment-2`

```
import math
from matplotlib import pyplot as plt
import numpy as np


def parse_file_to_objects(filepath):
    objects = []
    with open(filepath, 'r') as file:
        for line in file:
            parts = line.strip().split()
            if len(parts) != 2:
                continue
            try:
                x = float(parts[0])
                y = float(parts[1])
                objects.append({'x': x, 'y': y})
            except ValueError:
                continue
    return objects


def plot_unclustered_data(g1,g2,g3):
    g1_x = [point['x'] for point in g1]
    g1_y = [point['y'] for point in g1]

    g2_x = [point['x'] for point in g2]
    g2_y = [point['y'] for point in g2]

    g3_x = [point['x'] for point in g3]
    g3_y = [point['y'] for point in g3]

    plt.figure(figsize=(10, 6))
    plt.scatter(g1_x, g1_y, label="Group 1", alpha=0.6)
    plt.scatter(g2_x, g2_y, label="Group 2", alpha=0.6)
    plt.scatter(g3_x, g3_y, label="Group 3", alpha=0.6)
    plt.title("Parsed Data from Files")
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.legend()
    plt.grid(True)
    plt.show()


def plot_original_vs_clusters(g1, g2, g3, clusters):
    g1_x = [point['x'] for point in g1]
    g1_y = [point['y'] for point in g1]
    g2_x = [point['x'] for point in g2]
    g2_y = [point['y'] for point in g2]
    g3_x = [point['x'] for point in g3]
    g3_y = [point['y'] for point in g3]

    colors = [
        (0.1216, 0.4667, 0.7059, 1.0), (1.0, 0.498, 0.0549, 1.0), (0.1725, 0.6275, 0.1725,
1.0), (0.8392, 0.1529, 0.1569, 1.0),
        (0.5804, 0.4039, 0.7412, 1.0), (0.549, 0.3373, 0.2941, 1.0), (0.8902, 0.4667, 0.7608,
1.0), (0.498, 0.498, 0.498, 1.0),
        (0.7373, 0.7412, 0.1333, 1.0), (0.0902, 0.7451, 0.8118, 1.0), (0.6824, 0.7804, 0.9098,
 1.0), (1.0, 0.7333, 0.4706, 1.0),
        (0.5961, 0.8745, 0.5412, 1.0), (1.0, 0.5961, 0.5882, 1.0), (0.7725, 0.6902, 0.8353,
1.0), (0.7686, 0.6118, 0.5804, 1.0),
        (0.9686, 0.7137, 0.8235, 1.0), (0.7804, 0.7804, 0.7804, 1.0), (0.902, 0.902, 0.498,
1.0), (0.5019, 0.8509, 0.8784, 1.0),
        (0.1922, 0.2118, 0.5843, 1.0), (0.2706, 0.4588, 0.7059, 1.0), (0.4549, 0.6784, 0.8196,
 1.0), (0.6196, 0.7922, 0.8824, 1.0),
        (0.7765, 0.8588, 0.9373, 1.0), (0.902, 0.9294, 0.9686, 1.0), (0.1725, 0.2392, 0.1725,
1.0), (0.2549, 0.4275, 0.2549, 1.0),
        (0.4353, 0.6627, 0.4353, 1.0), (0.5961, 0.7686, 0.5961, 1.0), (0.7765, 0.8588, 0.7765,
 1.0), (0.902, 0.9294, 0.902, 1.0),
        (0.349, 0.2078, 0.2314, 1.0), (0.549, 0.4275, 0.451, 1.0), (0.7686, 0.651, 0.6667,
1.0), (0.8588, 0.7765, 0.7843, 1.0),
        (0.9373, 0.902, 0.9059, 1.0), (0.9882, 0.9647, 0.9647, 1.0), (0.2863, 0.2902, 0.1961,
1.0), (0.4549, 0.4588, 0.2863, 1.0)
    ]

    clustered_points = [np.array(c) for c in clusters]

    fig, axs = plt.subplots(1, 2, figsize=(14, 6))

    axs[0].scatter(g1_x, g1_y, label="Group 1", color=colors[2 % 10], alpha=0.6)
    axs[0].scatter(g2_x, g2_y, label="Group 2", color=colors[1 % 10], alpha=0.6)
    axs[0].scatter(g3_x, g3_y, label="Group 3", color=colors[0 % 10], alpha=0.6)
    axs[0].set_title("Original Groups")
    axs[0].set_xlabel("X")
    axs[0].set_ylabel("Y")
    axs[0].legend()
    axs[0].grid(True)

    for i, cluster in enumerate(clustered_points):
        axs[1].scatter(cluster[:, 0], cluster[:, 1],
                       color=colors[i], alpha=0.6, label=f"Cluster {i+1}")
    axs[1].set_title("Clusters from Divisive Clustering")
    axs[1].set_xlabel("X")
    axs[1].set_ylabel("Y")
    #axs[1].legend()
    axs[1].grid(True)

    plt.tight_layout()
    plt.show()



def dissimilarity(point1, point2):
    return math.dist(point1, point2)

def average_dissimilarity(point,data):

    average = 0

    for i in range(0, len(data)):

        average += dissimilarity(point,data[i])

    return average / len(data)



def divisive_clustering(cluster):
    #Step 2: Select the data point with the greatest average disimilarity and place it as the
first instance in a new cluster
    avg_data = list(map(lambda x: average_dissimilarity(x, cluster), cluster)) #
average_dissimilarities(cluster,cluster)
    idx = np.argmax(avg_data)

    new_cluster = [cluster[idx]]
    cluster = np.delete(cluster, idx, axis=0)
    updated_cluster = []

    transfers = 0
    #Step 3: Reassign elements from one cluster to the other,
    #       as long as their average dissimilarity to the first
    #       is greater that their distance to the new one
    for i in range(0,len(cluster)):
        avg_new_cluster = average_dissimilarity(cluster[i],new_cluster)
        avg_updated_cluster = average_dissimilarity(cluster[i],cluster)
        if avg_updated_cluster > avg_new_cluster:
            new_cluster.append(cluster[i])
            transfers += 1
            else:
                updated_cluster.append(cluster[i])

        return [updated_cluster, new_cluster, transfers]



    def cluster_data(data):
        #Step 1: Place all data in one cluster
        c0 = data
        clusters = [c0]

        #Step 4: Repeat the same process until all cluster have only one data point,
        #        or until no more transfers are possible.
        transfers = -1
        while not all(len(cluster) == 1 for cluster in clusters) and not transfers==0:
            largest_cluster_index = np.argmax([len(c) for c in clusters])
            updated_cluster, new_cluster, t = divisive_clustering(clusters[largest_cluster_index])

            clusters[largest_cluster_index] = updated_cluster
            clusters.append(new_cluster)
            transfers = t

        return clusters


    group1 =  parse_file_to_objects("om1.txt")
    group2 =  parse_file_to_objects("om2.txt")
    group3 =  parse_file_to_objects("om3.txt")
    all_points = group1 + group2 + group3
    data = np.array([[p['x'], p['y']] for p in all_points])

    clusters = cluster_data(data)
    plot_original_vs_clusters(group1,group2,group3,clusters)
```

The method **dissimilarity(point1, point2)** is used to calculate the disimilarity between two data points using their Eulidean distance

The method **average_dissimilarity(point,data)** takes as input a data point and an array of data and calculates its average disimilarity with the data by summing its dissimilarity with each specific array element and dividing by the number of elements in the array

The method **divisive_clustering(cluster)** contains the main implementation of the divisive clutering algorithm. It takes as argument a cluster, finds the data points with the greatest average disimilarity with all the other data points in the cluster, removes it from the cluster and adds it as the sole element to a new one. Then begins transfering elements from the old cluster to the new one with the condition that their average distance is greater to the old one than the new one. It returns the updated old cluster, the new cluster and the number of elements that were transfered from the old one to the new one.

The method **cluster_data(data)** takes the data array as a parameter, places initially all data in one cluster and then begins the Divisive clustering iteratively, until every cluster contains one element or no more transfers are possible. **Note that in each iteration the cluster selected to be divided is the largest one**.

Below you will find a plot of the original data grouped according to the dataset file that were obtained from (om1.txt $\rightarrow$ group1, om2.txt $\rightarrow$ group2, om3.txt $\rightarrow$ group3) and a plot of the clustered data after using the divisive clustering technique. In the first one each group has a different color, while in the second one every cluster has a separate color.

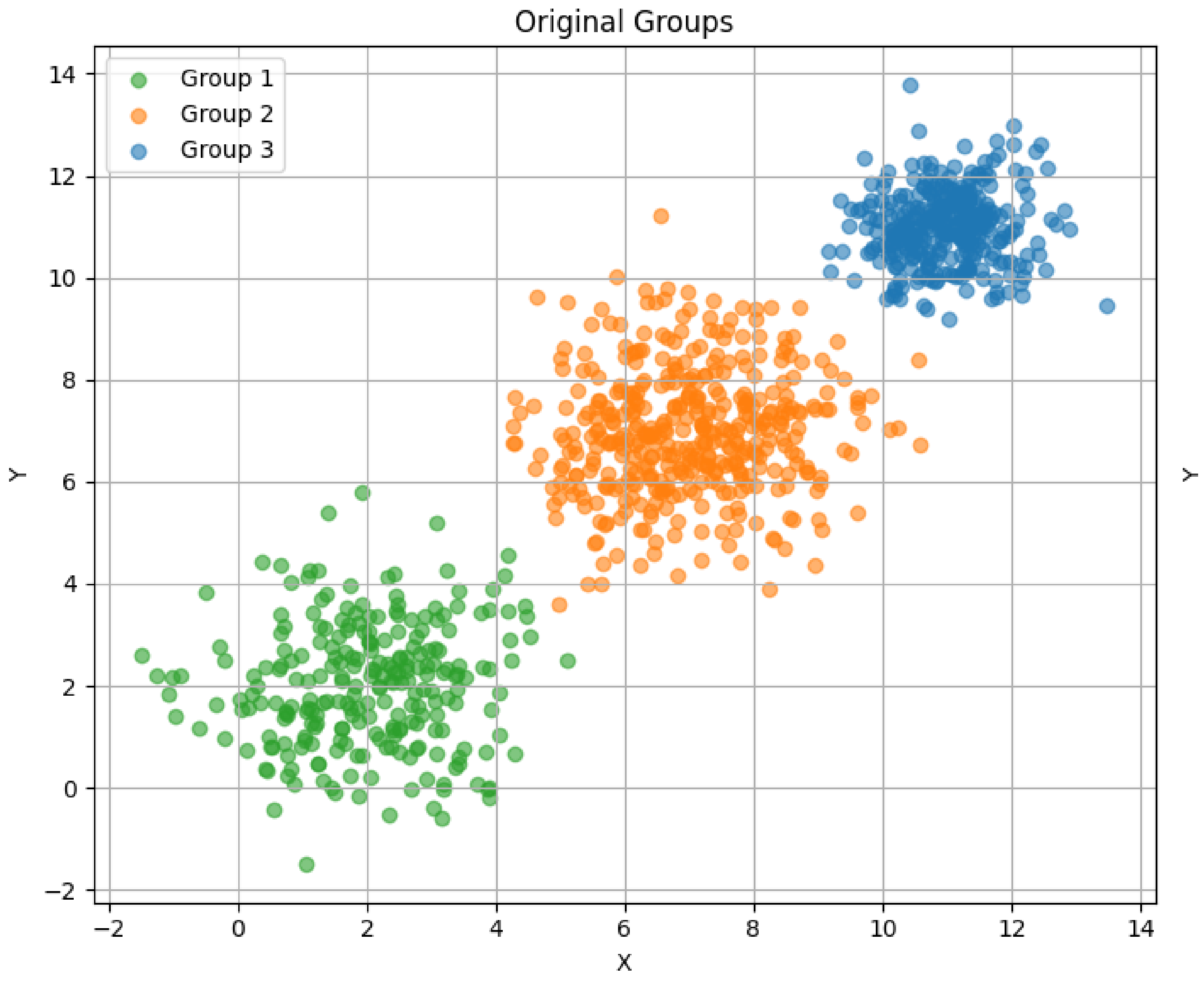


Figure 1: Original data groups as obtained from the respective dataset files

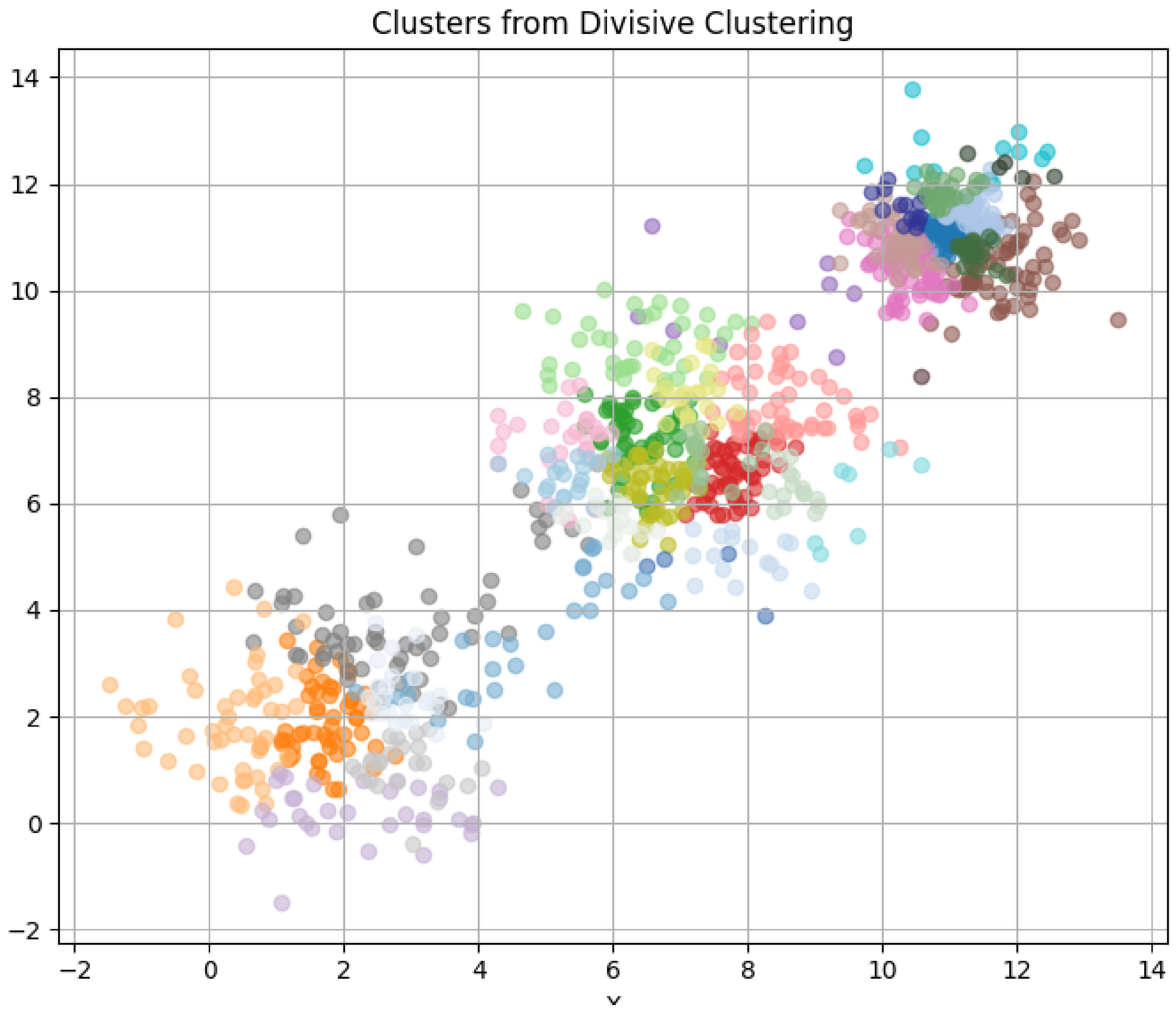


Figure 2: Clustered data after using the divisive clustering method